Non-clairvoyant reduction algorithms for heterogeneous platforms

Anne Benoit, Louis-Claude Canon, Loris Marchal

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **00**:1–13
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Non-clairvoyant reduction algorithms for heterogeneous platforms†

A. Benoit[1], L.-C. Canon[2], L. Marchal[1,*]

1. *LIP, École Normale Supérieure de Lyon, CNRS & INRIA, France,*
[anne.benoit,loris.marchal]@ens-lyon.fr

2. *FEMTO-ST, Université de Franche-Comté, Besançon, France,*
louis-claude.canon@univ-fcomte.fr

SUMMARY

We revisit the classical problem of the reduction collective operation in a heterogeneous environment. We discuss and evaluate four algorithms that are non-clairvoyant, i.e., they do not know in advance the computation and communication costs. On the one hand, **Binomial-stat** and **Fibonacci-stat** are static algorithms that decide in advance which operations will be reduced, without adapting to the environment; they were originally defined for homogeneous settings. On the other hand, **Tree-dyn** and **Non-Commut-Tree-dyn** are fully dynamic algorithms, for commutative or non-commutative reductions. We show that these algorithms are approximation algorithms with constant or asymptotic ratios. We assess the relative performance of all four non-clairvoyant algorithms with heterogeneous costs through a set of simulations. Our conclusions hold for a variety of distributions. Copyright © 2014 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: scheduling; reduction; approximation algorithms; non-clairvoyant algorithms.

1. INTRODUCTION

Reduction is one of the most common collective operations, together with the broadcast operation. Contrarily to a broadcast, it consists in gathering and summarizing information scattered at different locations. A classical example is when one wants to compute the sum of (integer) values distributed over a network: each node owns a single value and can communicate with other nodes and perform additions to compute partial sums. The goal is to compute the sum of all values. Reductions have been used in distributed programs for years, and standards such as MPI usually include a "reduce" function together with other collective communications (see [1, 2] for experimental comparisons). Many algorithms have been introduced to optimize this operation on various platforms, with homogeneous [3] or heterogeneous communication costs [4, 5]. Recently, this operation has received more attention due to the success of the MapReduce framework [6, 7], which has been popularized by Google. The idea of MapReduce is to break large workloads into small tasks that run in parallel on multiple machines, and this framework scales easily to very large clusters of inexpensive commodity computers. In this framework, the computation is split into a series of phases consisting in map and reduce operations. We review similar algorithms and use cases in the related work section (Section 2).

*Correspondence to: loris.marchal@ens-lyon.fr
†A preliminary version of this work appeared in HeteroPar'2013.

Our objective in this paper is to compare the performance of various algorithms for the reduce operations in a non-clairvoyant setting, i.e., when the algorithms are oblivious to the communication and computation costs (the time required to communicate or compute). This models well the fact that communication times cannot usually be perfectly predicted, and may vary significantly over time. We would like to assess how classical static algorithms perform in such settings, and to quantify the advantage of dynamic algorithms (if any). We use both theoretical techniques (worst-case analysis) and simulations on a wide range of random distributions.

The rest of the paper is organized as follows. Section 2 reviews existing reduction algorithms and other related work. Section 3 and 4 describes four algorithms and shows that they are approximation algorithms. Section 5 presents simulated executions of the previous algorithms and compares their respective performance, using several different random distributions for costs. Finally, we conclude and discuss future research directions in Section 6.

2. RELATED WORK

The literature has first focused on a variation of the reduction problem, the (global) combine problem [8, 9, 10]. Algorithmic contributions have then been proposed to improve MPI implementations and existing methods have been empirically studied in this context [11, 12]. Recent works concerning MapReduce either exhibit the reduction problem or highlight the relations with MPI collective functions. We describe below the most significant contributions.

Bar-Noy et al. [13] propose a solution to the global combine problem: similarly to allreduce, all machines must know the final result of the reduction. They consider the postal model with a constraint on the number of concurrent transfers to the same node (multi-port model). However, the postal model does not capture varying degree of overlapping between computations and communications.

Rabenseifner [3] introduces the *butterfly algorithm* for the same problem, with arbitrary array sizes. Several vectors must be combined into a single one by applying an element-wise reduction. Another solution has also been proposed when the number of machines is not a power of two [14]. These approaches are specifically adapted for element-wise reduction of arrays. Van de Geijn [15] also proposes a method with a similar cost. In our case, the reduction is not applied on an array and the computation is assumed to be indivisible.

Sanders et al. [16] exploit in and out bandwidths. Although the reduction does not require to be applied on arrays, the operation is split in at least two parts. This improves the approach based on a binary tree by a factor of two.

Legrand et al. [5] study steady-state situations where a series of reductions are performed. As in our work, the reduction operation is assumed to be indivisible, transfers and computations can overlap and the full-duplex one-port model is considered. The solution is based on a linear program and produces asymptotically optimal schedules with heterogeneous costs.

Liu et al. [4] propose a 2-approximation algorithm for heterogeneous costs and non-overlapping transfers and computations. Additionally, they solve the problem when there are only two possible speeds or when any communication time is a multiple of any shorter communication time. In the homogeneous case, their solution builds binomial trees, which are covered in Section 3.

In the MPI context, Kielmann et al. [17] design algorithms for collective communications, including `MPI_Reduce`, in hierarchical platforms. They propose three heuristics: flat tree for short messages, binomial tree for long messages, and a specific procedure for associative reductions in which data are first reduced locally on each cluster before the results are sent to the root process. Pjesivac-Grbovic et al. [1] conduct an empirical and analytical comparison of existing heuristics for several collective communications. The analytical costs of these algorithms are first determined using different classical point-to-point communication models, such as Hockney, LogP/LogGP and PLogP. The compared solutions are: flat tree, pipeline, binomial tree, binary tree, and k-ary tree. Thakur et al. [2] perform a similar study for several MPI collective operations and compare the binomial tree with the butterfly algorithm [3] for `MPI_Reduce`. These works, however, do not provide any guarantee on the performance.

DOI: 10.1002/cpe

Finally, this problem has also been addressed for MapReduce applications. Agarwal et al. [18] present an implementation of allreduce on top of Hadoop based on spanning trees. Moreover, some MapReduce infrastructures, such as MapReduce-MPI[†], are based on MPI implementations and benefit from the improvements done on `MPI_Reduce`.

The design and the analysis of algorithms in dynamic context has already received some attention. The closest related work is probably [19], in which the authors study the robustness of several task-graph scheduling heuristics for building static schedules. The schedules are built with deterministic costs and the performance is measured using random costs. [20] studies the problem of computing the average performance of a given class of applications (streaming applications) in a probabilistic environment. With dynamic environments comes the need for robustness to guarantee that a given schedule will behave well in a disturbed environment. Among others, [21] studies and compares different robustness metrics for makespan/reliability optimization on task-graph scheduling. Optimizing the performance for task-graph scheduling in dynamic environments is a natural follow-up, and has been tackled notably using the concepts of IC (Internet-based Computing) and area-maximizing schedules [22].

3. MODEL AND ALGORITHMS

We consider a set of n processors (or nodes) $P_0, \ldots, P_{n-1}$, and an associative operation $\oplus$. Each processor P_i owns a value v_i. The goal is to compute the value $v = v_0 \oplus v_1 \oplus \cdots \oplus v_{n-1}$ as fast as possible, i.e., to minimize the total execution time to compute the reduction. We do not enforce a particular location for the result: at the end of the reduction, it may be present on any node.

There are two versions of the problem, depending on whether the $\oplus$ operation is commutative or not. For example, when dealing with numbers, the reduction operation (sum, product, etc.) is usually commutative while some operations on matrices (such as the product) are not. The algorithms proposed and studied below deal with both versions of the problem.

We denote by $d_{i,j}$ the time needed to send one value from processor P_i to processor P_j. A value may be an initial value or a partial result. When a processor P_i receives a value from another processor, it immediately computes the reduction with its current value. To model communication congestion, we assume that each processor can receive at most one result at a time. Without this assumption, we may end up with unrealistic communication schemes where all but one processors simultaneously send their value to a single target processor, which would exceed the capacity of the incoming communication port on the target processor. The communication costs are heterogeneous, that is we may well have different communication costs depending on the receiver ($d_{i,j} \neq d_{i,j'}$), on the sender ($d_{i,j} \neq d_{i',j}$) and non-symmetric costs ($d_{i,j} \neq d_{j,i}$). Even though these costs are fixed, we consider non-clairvoyant algorithms that make decisions without any knowledge of these costs.

The computation time of the atomic reduction on processor P_i is denoted by c_i. In the case of a non-commutative operation, we ensure that a processor sends its value only to a processor that is able to perform a reduction with its own value. Formally, assume that at a given time, a processor owns a value that is the reduction of $v_i \oplus \cdots \oplus v_j$, which we denote by $[v_i, v_j]$; the processor may only send this value to a processor owning a value $[v_k, v_{i-1}]$ or $[v_{j+1}, v_k]$, which is called a *neighbor value* in the following.

During a reduction operation, a processor sends its value at most once, but it may receive several values. It computes a partial reduction each time it receives a value. Thus, the communication graph of a reduction is a tree (see Figure 1): the vertices of the tree are the processors and its edges are the communications of values (initial or partially reduced values). In the example, P_0 receives the initial value from P_1, and then a partially reduced value from P_2. In the following, we sometimes identify a reduction algorithm with the tree it produces.

We now present the four algorithms that are studied in this paper. The first two algorithms are static algorithms, i.e., the tree is built before the actual reduction. Thus, they may be applied for

[†] http://www.sandia.gov/~sjplimp/mapreduce.html

DOI: 10.1002/cpe

commutative or non-commutative reductions. The last two algorithms are dynamic: the tree is built at run-time and depends on the actual durations of the operations.

The first algorithm, called **Binomial-stat**, is organized with $\lceil \log_2 n \rceil$ rounds. Each round consists in reducing a pair of processors that own a temporary or initial data using a communication and a computation. During round $k = 1, \ldots, \lceil \log_2 n \rceil$, each processor $i2^k + 2^{k-1}$ ($i = 0, \ldots, 2^{\lceil \log_2 n \rceil - k} - 1$) sends its value to processor $i2^k$, which reduces it with its own value: at most $2^{\lceil \log_2 n \rceil - k + 1}$ processors are involved in round k. Note that rounds are not synchronized throughout the platform: each communication starts as soon as the involved processors are available and have terminated the previous round. We can notice that the communication graph induced by this strategy is a binomial tree [23, Chapter 19], hence the name of the algorithm. This strategy is illustrated on Figure 2(a).

The second algorithm, called **Fibonacci-stat**, is constructed in a way similar to Fibonacci numbers. The schedule constructed for order k, denoted by FS_k ($k > 0$), first consists in two smaller order schedules FS_{k-1} and FS_{k-2} put in parallel. Then, during the last computation of FS_{k-1}, the root of FS_{k-2} (that is, the processor that owns its final value) sends its value to the root of FS_{k-1}, which then computes the last reduction. A schedule of order -1 or 0 contains a single processor and no operation. This process is illustrated on Figure 2(b). Obviously, the number of processors involved in such a schedule of order k is F_{k+2}, the $(k+2)$th Fibonacci number. When used with another number n of processors, we compute the smallest order k such that $F_{k+2} \geq n$ and use only the operations corresponding to the first n processors in the schedule of order k.

The previous two schedules were proposed in [24], where their optimality is proved for special homogeneous cases: **Binomial-stat** is optimal both when the computations are negligible in front of communications ($c_i = 0$ and $d_{i,j} = d$) and when the communications are negligible in front of computations ($c_i = c$ and $d_{i,j} = 0$). **Fibonacci-stat** is optimal when computations and communications are equivalent ($c_i = c = d_{i,j} = d$). In the non-commutative case, both algorithms build a tree such that only neighboring partial values are reduced. In the commutative case, any permutation of processors can be chosen.

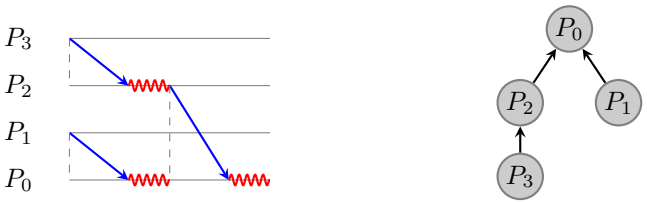


Figure 1. Schedule and communication graph for reducing four values. Blue arrows represent communications while red springs stand for computations.

(a) **Binomial-stat** (b) **Fibonacci-stat**

Figure 2. Schedules for **Binomial-stat** of order 3 and **Fibonacci-stat** of order 4, both using 8 processors. For **Fibonacci-stat**, the two schedules of order 2 and 3 used in the recursive construction are highlighted in green and red.

DOI: 10.1002/cpe

Then, we move to the design of dynamic reduction algorithms, i.e., algorithms that take communication decisions at runtime. The first dynamic algorithm, called **Tree-dyn**, is a simple greedy algorithm. It keeps a slot (initially empty), and when a processor is idle, it looks into the slot. If the slot is empty, the processor adds its index in the slot, otherwise it empties the slot and starts a reduction with the processor that was in the slot (i.e., it sends its value to the processor that was in the slot, and the latter then computes the reduced value). It means that a reduction is started as soon as two processors are available. Since in the obtained reduction tree, any two processors may be paired by a communication, this can only be applied to commutative reductions.

Finally, **Non-Commut-Tree-dyn**, is an adaptation of the previous dynamic algorithm to non-commutative reductions. In this algorithm, when a processor is idle, it looks for another idle processor with a neighbor value (as described above). Now, we keep an array of idle processors rather than a single slot. If there is an idle neighbor processor, a communication is started between them, otherwise the processor waits for another processor to become idle.

4. WORST-CASE ANALYSIS FOR COMMUTATIVE REDUCTIONS

We analyze the commutative algorithms in the worst case, and we provide some approximation ratios, focusing on communication times. We let $d = \min_{i,j} d_{i,j}$, $D = \max_{i,j} d_{i,j}$, $c = \min_i c_i$, and $C = \max_i c_i$.

Let us first recall results from [24], in the context of identical communication costs ($d = D$) and identical computation costs ($c = C$). The duration of the schedule built by **Fibonacci-stat** at order k is $d + (k-1)\max(d,c) + c$, and the number of reduced elements n is such that $F_{k+1} < n \leq F_{k+2}$, where F_k is the kth Fibonacci number.

In order to ease the following worst-case analysis, we first derive bounds on the minimum duration of any schedule depending on the number of elements to reduce and the minimum costs (n, c and d).

Lemma 1
The order k of a schedule built by **Fibonacci-stat** can be bounded as follows:

$$\frac{\log_2 n}{\log_2 \varphi} - 1 < k < \frac{\log_2 n}{\log_2 \varphi} + 1,$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio and n is the number of elements to reduce.

Proof
We know that $F_{k+1} < n \leq F_{k+2}$, where F_k is the kth Fibonacci number. By definition of the Fibonacci numbers, we have $F_k = \frac{1}{\sqrt{5}}(\varphi^k - (1-\varphi)^k)$. Therefore,

$$\frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^{k+1}) < n \leq \frac{1}{\sqrt{5}}(\varphi^{k+2} - (1-\varphi)^{k+2}).$$

Since $-1 < 1-\varphi < 0$, it follows that

$$\frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^2) < n < \frac{1}{\sqrt{5}}(\varphi^{k+2} - (1-\varphi)^3)$$

as soon as $k \geq 1$. We therefore have $\varphi^{k+1} < \sqrt{5}n + (1-\varphi)^2$ and $\sqrt{5}n + (1-\varphi)^3 < \varphi^{k+2}$.
We easily derive the following inequalities:

$$\frac{\log_2 n}{\log_2 \varphi} + \underbrace{\frac{\log_2\left(\sqrt{5} + \frac{(1-\varphi)^3}{n}\right)}{\log_2 \varphi} - 2}_{\geq -1 \text{ when } n \geq 1} < k < \frac{\log_2 n}{\log_2 \varphi} + \underbrace{\frac{\log_2\left(\sqrt{5} + \frac{(1-\varphi)^2}{n}\right)}{\log_2 \varphi} - 1}_{\leq 1 \text{ when } n \geq 1}.$$

Finally, $\frac{\log_2 n}{\log_2 \varphi} - 1 < k < \frac{\log_2 n}{\log_2 \varphi} + 1$. □

DOI: 10.1002/cpe

Proposition 1
The time taken to reduce n elements using any reduction algorithm is larger than $\max(c,d)\lceil\log_2 n\rceil$ and than $\min(c,d)\frac{\log_2 n}{\log_2 \varphi}$.

Proof
First, any reduction algorithm has a duration that is lower bounded by the duration of the **Binomial-stat** algorithm on a scenario where each communication costs d and each computation costs c. Because of the overlap between communication and computation, each step of this reduction has a duration of $\max(c,d)$, and the total duration of this reduction is $\max(c,d)\lceil\log_2(n)\rceil$, hence the first bound.

Second, the optimal schedule when all costs (communication and computation) are equal to $\min(c,d)$ is obtained with **Fibonacci-stat**, and its length is greater than $(k+1)\min(c,d)$. Using Lemma 1, we obtain the second lower bound on the schedule duration. □

We are ready to state the approximation ratios for **Binomial-stat** and **Tree-dyn**. Recall that a λ-approximation algorithm is an algorithm whose execution time is polynomial in the instance size, and that returns an approximate solution, guaranteed to be, in the worst case, at a factor λ away from the optimal solution.

Theorem 1
Binomial-stat and **Tree-dyn** are $\frac{C+D}{\max(c,d)}$-approximation algorithms, and this ratio can be achieved when $\min(c,d)=0$.

Additionally, these algorithms are also $\frac{C+D}{\min(c,d)}\left(1+\frac{1}{\log_2 n}\right)\log_2\varphi$-approximation algorithms, where $\varphi=\frac{1+\sqrt{5}}{2}$ is the golden ratio ($\log_2\varphi\approx 0.69$), and this ratio can be achieved when $c=d$.

Proof
The proof is in two steps: we first prove the approximation ratios by exhibiting an upper bound on the duration of **Binomial-stat** and **Tree-dyn**, before showing scenarios that achieve these ratios.

Let us consider an algorithm that would perform reductions through $\lceil\log_2 n\rceil$ synchronized steps. At each step, all processors possessing an element group themselves in pair for sending and reducing their elements (they perform their reductions pairwise). At each step, half the elements are processed. This algorithm takes at most a time $(C+D)\lceil\log_2 n\rceil$, because, in the worst case, one computation and one communication at each step may be of maximum duration. **Binomial-stat** is faster than this algorithm because it has the same structure for the reductions (the source and destination of each communication is the same), except that no synchronization is required between each global step. Indeed, communications happen as soon as possible. **Tree-dyn** is also faster than this algorithm because two available processors may start as soon as possible without delaying the remaining steps. Therefore, the time taken by **Binomial-stat** and **Tree-dyn** is not greater than $(C+D)\lceil\log_2 n\rceil$. The comparison of this upper bound to each of the two lower bounds given by Proposition 1 gives two approximation ratios. The first approximation ratio is $\frac{C+D}{\max(c,d)}$. The second ratio writes $\frac{(C+D)\lceil\log_2 n\rceil}{\min(c,d)\log_2 n/\log_2\varphi}\leq\left(\frac{C+D}{\min(c,d)}\log_2\varphi\right)\left(1+\frac{1}{\log_2 n}\right)$.

We now exhibit a scenario for which these ratios are achieved. Let $d_{i,j}=d$ and $d_{j,i}=D$, for all $i<j$, $c_i=C$ when i is even and $c_i=c$ if i is odd. With both **Binomial-stat** and **Tree-dyn**, we consider that any processor P_i sends its element to a processor P_j such that $i>j$, which takes a total time $(C+D)\lceil\log_2 n\rceil$. Indeed, we consider that during the first step, each machine with an odd index sends an element to a machine with an even index, and these latter machines are performing the computations. The optimal solution performs only computations of size c and communications of size d (from P_i to P_j, with $j>i$, and only machines with odd index perform computations).

When $\min(c,d)=0$, the optimal total time is $\max(c,d)\lceil\log_2 n\rceil$ and the first ratio is achieved. With $c=d$, a solution with a Fibonacci tree using only small communications is optimal [24]; furthermore, it completes in time $(k+1)d$. By Lemma 1, we have $k>\frac{\log_2 n}{\log_2\varphi}-1$ and the ratio is $\frac{C+D}{d/\log_2\varphi}\frac{\lceil\log_2 n\rceil}{\log_2 n}$, which concludes the proof. □

DOI: 10.1002/cpe

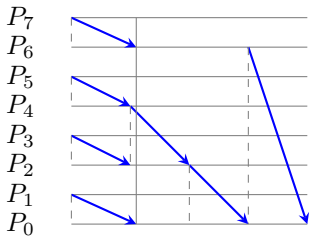


Figure 3. Schedule of a worst-case scenario for **Non-Commut-Tree-dyn**.

Unfortunately, the first approximation ratio of the previous theorem does not hold for **Fibonacci-stat** and **Non-Commut-Tree-dyn**:

- For **Fibonacci-stat**, consider the schedule in Figure 2(b). Even without computation costs, the number of communication steps is 4 because P_0 has to receive four messages and these communications must be sequential. If these four communications take a time D (and all other $d_{i,j}$'s are d), the optimal solution may complete in a time $3d$ (see for instance Figure 2(a), for **Binomial-stat**), hence a ratio of $\frac{4}{3}\frac{D}{d}$. We prove below (see Theorem 2) that **Fibonacci-stat** is asymptotically a $\frac{\max(C,D)}{\max(c,d)}\frac{1}{\log_2\varphi}$-approximation, where $\frac{1}{\log_2\varphi} \approx 1.44$.
- For **Non-Commut-Tree-dyn**, the number of steps may well exceed $\lceil \log_2(n) \rceil$ as well. Consider for instance that for a reduction on 8 processors, in the first step neighboring processors communicate together two by two (as depicted on Figure 3). However, P_4 and P_2 terminate slightly before P_6 and P_0, and initiate a communication. Then P_6 and P_0 must wait until completion of this partial reduction, and then two more steps are required to complete the reduction. Here again, the worst-case ratio is $\frac{4}{3}\frac{D}{d}$ (with $d_{i,j} = D$ if $i > j$, and $d_{i,j} = d$ otherwise). We do not prove any approximation ratio for this algorithm, because it does not seem very fair to compare a non-commutative version of the algorithm with the optimal commutative solution.

Theorem 2
Fibonacci-stat is a $\frac{\max(C,D)}{\max(c,d)}\frac{1}{\log_2\varphi} + \frac{C+D}{\max(c,d)}\frac{1}{\lceil\log_2 n\rceil}$-approximation algorithm, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio $(1/\log_2\varphi \approx 1.44)$.

Additionally, this algorithm is also a $\frac{\max(C,D)}{\min(c,d)}$-approximation.

Proof
The length of the **Fibonacci-stat** schedule in the worst case is bounded by $D + (k-1)\max(D,C) + C$, with k the order of the Fibonacci schedule [24]. By Lemma 1, this length is further bounded by $D + \frac{\log_2 n}{\log_2\varphi}\max(D,C) + C$. Recall, from Proposition 1, that $\max(c,d)\lceil\log_2 n\rceil$ is a lower bound on the reduction time, which gives the first approximation ratio.

The length of the **Fibonacci-stat** schedule in the worst case is also bounded by $(k+1)\max(D,C)$. The second lower bound of Proposition 1 is $(k+1)\min(d,c)$, which gives the second approximation ratio. □

Table I summarizes the ratios for the case when $c \leq d$ (the opposite case is symmetrical).

Note that for all algorithms, the second approximation ratio is asymptotically tighter when $1 \geq \frac{c}{d} \geq \log_2\varphi \approx 0.69$, which corresponds to the situation for which the overlap between computations and communications is significant, whereas the first ratio is tighter for a smaller $\frac{c}{d}$ ratio, corresponding to a low overlap between communications and computations.

Using these asymptotic ratios, one may wonder when **Fibonacci-stat** is expected to have a better performance than **Binomial-stat** and **Tree-dyn**. To this end, we consider the ratio between C and D that makes **Fibonacci-stat** the best strategy. This asymptotically occurs when $\min\left(\frac{C}{D}, \frac{D}{C}\right) \geq \frac{1}{\log_2\varphi} - 1 \approx 0.44$. This is the case when there is a significant overlap between computations and

DOI: 10.1002/cpe

Table I. Approximation ratios for commutative algorithms (we assume $c \leq d$ due to symmetry).

	first ratio (low overlap)	second ratio (high overlap)
Binomial-stat and **Tree-dyn**	$\frac{C+D}{d}$	$\frac{C+D}{c}\left(1+\frac{1}{\log_2 n}\right)\log_2\varphi$
Fibonacci-stat	$\frac{\max(C,D)}{d}\frac{1}{\log_2\varphi}+\frac{C+D}{d}\frac{1}{\lceil\log_2 n\rceil}$	$\frac{\max(C,D)}{c}$

communications (i.e., when the maximum computation cost is approximatively between 44% and 228% of the maximum communication cost).

Note also that when $c = d = C = D$, the asymptotic approximation ratio of **Binomial-stat** is $2\log_2\varphi \approx 1.39$, which nicely complements the asymptotic approximation ratio of $\frac{1}{\log_2\varphi} \approx 1.44$ for **Fibonacci-stat** in [24].

5. SIMULATION RESULTS

In the first part of this section, we consider that the d_{ij} and c_i costs are distributed according to a gamma distribution, which is a generalization of exponential and Erlang distributions. This distribution has been advocated for modeling job runtimes [25, 26]. This distribution is positive and it is possible to specify its expected value (μ_d or μ_c) and standard deviation (σ_d or σ_c) by adjusting its parameters. Further simulations with other distributions concur with the following observations and are presented at the end of this section. This suggests that our conclusions are not strongly sensitive to the distribution choice. All simulations were performed with an ad-hoc simulator on a desktop computer.

Cost dispersion effect. In this first simulation, we are interested in characterizing how the dispersion of the communication costs affects the performance of all methods. In order to simplify this study, no computation cost is considered ($\mu_c = 0$). The dispersion is defined through the coefficient of variation (CV), which is defined as the ratio of the standard deviation over the expected value (this latter is set to $\mu_d = 1$). The number of processors is $n = 64$ and the time taken by each method is measured over 1,000,000 Monte Carlo (MC) simulations (i.e., simulations have been performed with 1,000,000 distinct seeds).

On a global level, Figure 4 shows the expected performance of the four studied reduction algorithms with distinct CVs. When the heterogeneity is noticeable (CV greater than 1), the performance decreases significantly. In these cases, schedules are mostly affected by a few extreme costs whose importance depends on the CV. Additionally, the variability in the schedule durations is also impacted by the CV (i.e., two successive executions with the same settings may lead to radically different performance depending on the schedule).

Several observations can be made relatively to each method. As expected, **Binomial-stat** is similar to **Tree-dyn** for CVs lower than 0.1. In this case, the improvement offered by **Tree-dyn** may not outweigh the advantage of following a static plan in terms of synchronization. For CVs greater than 1, both static approaches perform equally with a similar dispersion. For all CVs, **Tree-dyn** has the best expected performance while **Fibonacci-stat** has the worst (which is expected since no computation cost is considered), and **Non-Commut-Tree-dyn** has the second best expected performance when the CV is greater than 0.3. Finally, when the CV is close to 10, all methods are equivalent as a single communication with a large cost may impact the entire schedule duration. In terms of robustness, we can see that **Fibonacci-stat** and **Non-Commut-Tree-dyn** are the two best methods for absorbing variations as their expected durations remain stable longer (until the CV reaches 0.3). This is due to the presence of idleness in their schedules that can be used when required.

Non-negligible computation costs. When considering nonzero computation costs, we reduce the number of parameters by applying the same CV to the computation and to the communication costs

DOI: 10.1002/cpe

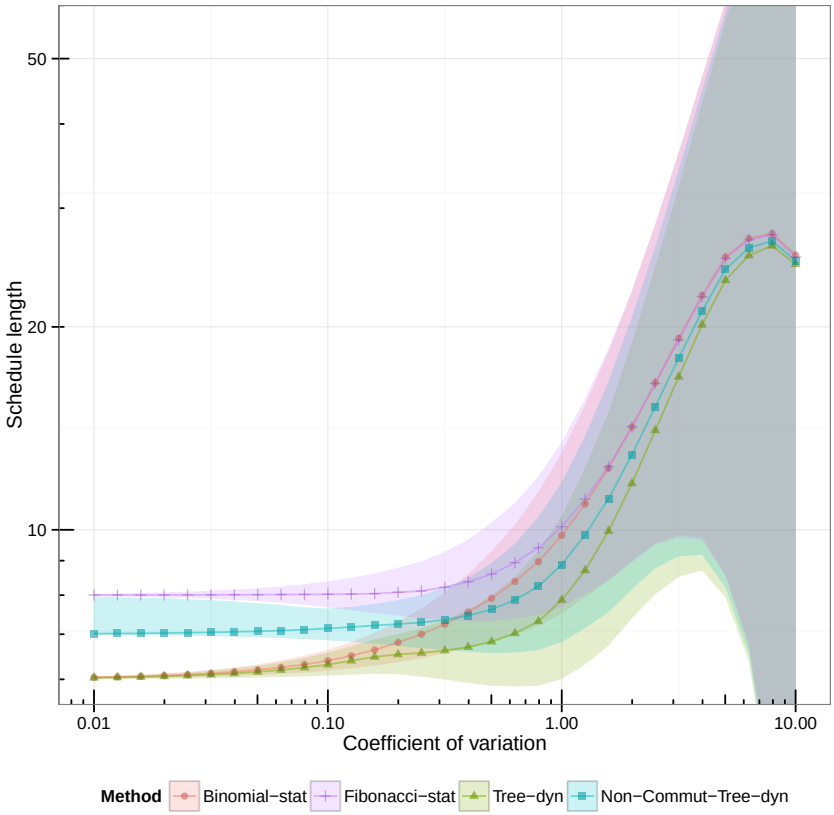


Figure 4. Average schedule length for each method over 1,000,000 MC simulations with $n = 64$, $\mu_d = 1$, $\mu_c = 0$, and varying coefficients of variation for the communication costs. The lower part of the ribbons corresponds to the 10% quantile while the upper part corresponds to the 90% quantile for each method.

(i.e., $\frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$). As **Fibonacci-stat** is designed for overlapping computations and communications, we characterize the cases when this approach outperforms **Tree-dyn**.

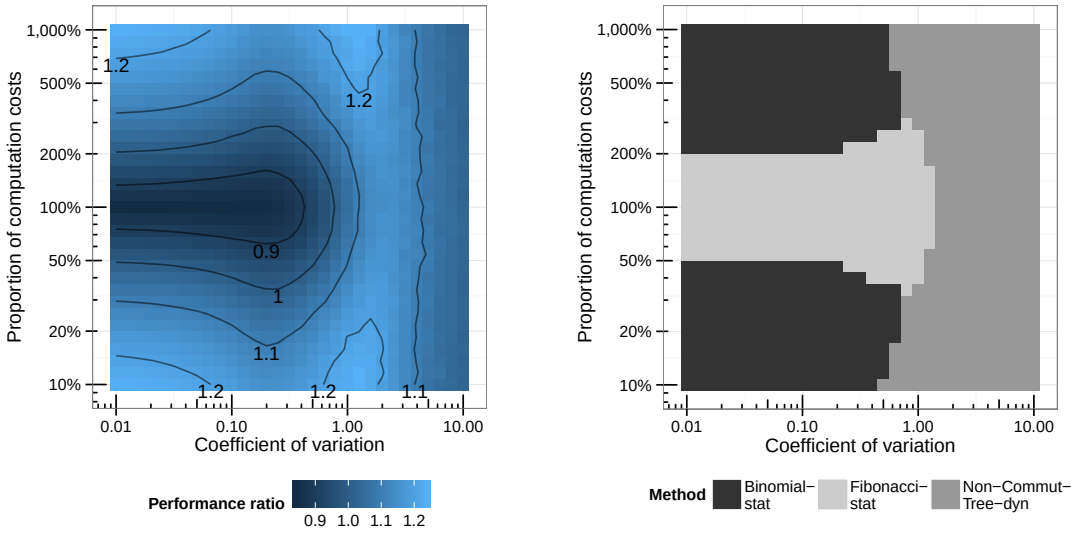
Figure 5(a) shows the improvement of **Tree-dyn** over **Fibonacci-stat** when varying the CV and the ratio $\frac{\mu_c}{\mu_d}$, which corresponds to the overlapping degree between computations and communications. The darker the color, the better is **Fibonacci-stat** compared to **Tree-dyn**. The contour line with value 1 delimits the dark area for which **Fibonacci-stat** is better than **Tree-dyn** on average. This occurs when the computation cost is greater than around half the communication cost and when the variability is limited. When the computation costs are low (proportion of computation costs of 10%, i.e., $\frac{\mu_c}{\mu_d} = 0.1$), the ratio evolution is consistent with the previous observations.

Figure 5(a) is horizontally symmetrical as any case such that $\frac{\mu_c}{\mu_d} > 1$ (proportion of computation costs greater than 100%) is equivalent to the situation where the communication and the computation costs are swapped (and for which $\frac{\mu_c}{\mu_d} < 1$). These costs can be exchanged because a communication is always followed by a reduction operation.

Non-Commutative Operation. Finally, we assess the performance of **Non-Commut-Tree-dyn** by comparing it to all other methods that support a non-commutative operation when varying the dispersion and the overlapping degree as in the previous study.

Figure 5(b) shows the method with the best average performance when varying the CV and the ratio $\frac{\mu_c}{\mu_d}$. We see that **Non-Commut-Tree-dyn** has the best performance when the cost dispersion is large. Additionally, the transition from **Binomial-stat** to **Fibonacci-stat** is when the computation cost reaches half the communication cost (as on Figure 5(a)). With low computation costs (proportion of computation costs of 10%), the results are also consistent with Figure 4.

DOI: 10.1002/cpe



(a) Non-negligible communication

(b) Non-commutative operation

Figure 5. Ratio of the average performance of **Fibonacci-stat** and **Tree-dyn** (a) and method with the best average performance (b) over 1,000 MC simulations for each square with $n = 64$, $\mu_d = 1$, CV$= \frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$, varying the coefficient of variation (CV) for the costs and varying $\frac{\mu_c}{\mu_d}$.

Distribution Effect. The probability distribution could impact the previous conclusions. To assess this influence, we repeated the experiments summarized by Figures 5(a) and 5(b) with other probability distributions.

Table II summarizes the tested distributions along with the corresponding parameters. While the CV varies from 0.01 to 10, the mean is always one. We discarded distributions with negative outcomes such as the normal distribution to have only positive costs. For some distributions, CVs greater than some threshold lead to non-positive support or to invalid parameters (the CV range is thus restrained for those distributions). Finally, we scale some distributions with two additional parameters to control the CV: m is an additive parameter (the minimum value in the case of the exponential distribution) and M is a multiplicative parameter (the maximum value in the case of the Bernoulli distribution).

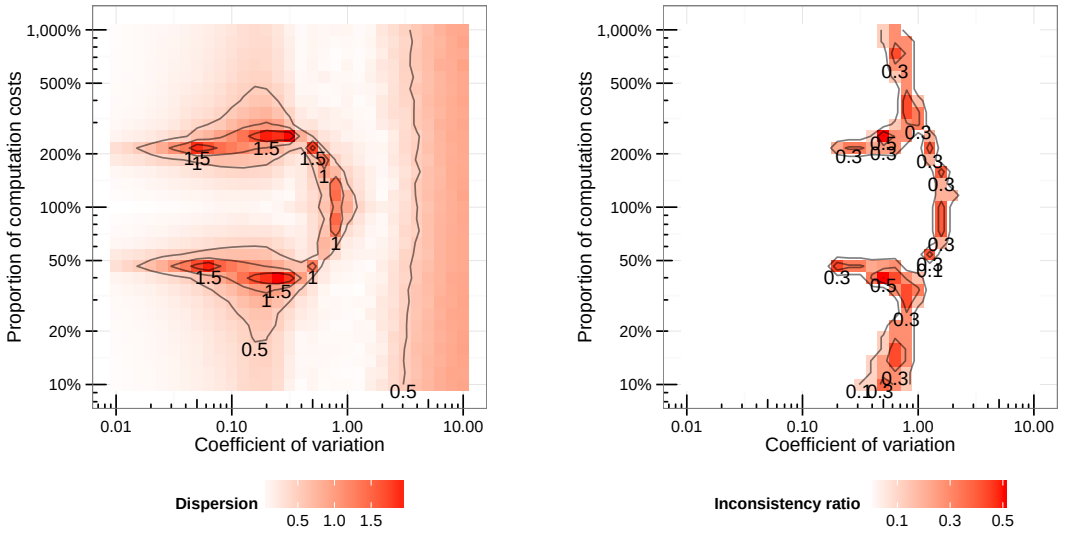
On Figure 6(a), the same results shown by Figure 5(a) were aggregated for all ten distributions given by Table II. For a given scenario, that is, a given CV and a given μ_c/μ_d value, we first compute the ratio of the average time for **Fibonacci-stat** over the average time for **Tree-dyn** (over 1,000 MC simulations), for all possible distributions. Our objective is to check whether these ten ratios r are close to each other, or if some distributions lead to inconsistent results. To do this, we first compute the range of the ratios, and we normalize this range by the maximum distance between a ratio and one:

$$dispersion = \frac{\max r - \min r}{\max |r - 1|}.$$

For instance, if all ratios are between 1.3 and 1.4, then the range is 0.1 and the maximum distance to 1 is 0.4, which gives a dispersion of 0.25. The rationale behind this normalization is to give a dispersion larger than 1 when results are inconsistent among all distributions, that is, if the **Fibonacci-stat** is faster than **Tree-dyn** for some distribution and slower for another distribution.

Dark zones on Figure 5(a) correspond to scenarios where there are significant differences between the different distributions. Note that our method of combining a range with a relative measure purposely provides a pessimistic value (it considers the two most extreme distributions). On the contrary, other metrics such as the standard deviation would hide the differences between most

DOI: 10.1002/cpe



(a) Non-negligible communication

(b) Non-commutative operation

Figure 6. Dispersion of the average relative performance of **Fibonacci-stat** and **Tree-dyn** (a) and dispersion of the method with the best average performance (b) over 1,000 MC simulations for each square with $n = 64$, $\mu_d = 1$, CV$= \frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$, varying the coefficient of variation (CV) for the costs, varying $\frac{\mu_c}{\mu_d}$ and with the ten distributions given by Table II.

distributions, which are small. Note that most of this figure is almost white, which means that the results are coherent using all distributions. There are mostly three areas where there is divergence:

Table II. Experimented probability distributions with their parameters given a mean fixed to one and a variable CV. Distributions are scaled with the additional parameters m and M (the outcome is incremented by m or multiplied by M). The CV is 0.5 for each example density.

Name	Parameters	CV range	Density
Bernoulli	$p = \frac{1}{M}$, $M = 1 + CV^2$	$[0; \infty[$	
Beta (0.01)	$\alpha = 0.01$, $\beta = \alpha(M-1)$, $M = \frac{1+CV^2}{1-\alpha CV^2}$	$[0; 10[$	
Beta (1)	$\alpha = 1$, idem for β and M	$[0; 1[$	
Beta (100)	$\alpha = 100$, idem for β and M	$[0; 0.1[$	
Binomial	$n = 100$, $p = \frac{1}{1+nCV^2}$, $M = \frac{1}{np}$	$[0; \infty[$	
Exponential	$\lambda = \frac{1}{CV}$, $m = \frac{1}{1-CV}$	$[0; \infty[$	
Gamma	shape $=$ rate $= \frac{1}{CV^2}$	$[0; \infty[$	
Poisson	$\lambda = CV^2$, $m = 1 - \lambda$	$[0; 1]$	
Triangle	$\min = 1 - \sqrt{6}CV$, $\max = 1 + \sqrt{6}CV$	$[0; \frac{1}{\sqrt{6}} \approx 0.41]$	
Uniform	$\min = 1 - \frac{\sqrt{12}}{2}CV$, $\max = 1 + \frac{\sqrt{12}}{2}CV$	$[0; \frac{2}{\sqrt{12}} \approx 0.58]$	

DOI: 10.1002/cpe

when the CV is around 0.25 with 40% of computation (and its symmetric counterpart with 250% of computation); when the CV is around 0.8 with perfect overlapping; and when the CV is greater than 5. The first two areas are related to the proximity between both methods, hence a slight quantitative performance difference is amplified by the relative metric that is used (these transitions between both methods occur for the same set of parameters however). The last area is essentially due to the exponential method for which the improvement of the dynamic method is stronger than with other distributions. On overall, all distributions show significant quantitative consistency (95% of the dispersions are below 1 and the median dispersion is 0.27).

Figure 6(b) aggregates results shown by Figure 5(b) for all distributions given by Table II. For a given scenario, we compute the expected best method for each distribution (using 1,000 MC simulations). The "inconsistency ratio" on Figure 6(b) is the proportion of the distributions for which the expected best method differs from the most frequent one. That is, an inconsistency ratio of 0.3 means that the best method for 3 out of the 10 distributions differs from the most frequent best method (which appears in the other 7 distributions). Once again, there is very little difference between the different distributions, and the discrepancies are restricted to the frontiers defining the zones where one strategy dominates the others.

In conclusion, using different distributions yields quantitative and qualitative consistent results, which suggests that our conclusions stand independently of the distribution.

6. CONCLUSION

In this paper, we have studied the problem of performing a non-clairvoyant reduction on a distributed heterogeneous platform. Specifically, we have compared the performance of traditional static algorithms, which build an optimized reduction tree beforehand, against dynamic algorithms, which organize the reduction at runtime. Our study includes both commutative and non-commutative reductions. We have first proposed approximation ratios for all commutative algorithms using a worst-case analysis. Then, we have evaluated all algorithms through extensive simulations using different random distributions to show when dynamic algorithms become more interesting than static ones. We have outlined that dynamic algorithms generally achieve better makespan, except when the heterogeneity is limited and for specific communication costs (no communication cost for **Binomial-stat**, communication costs equivalent to computation costs for **Fibonacci-stat**). The worst-case analysis has also confirmed this last observation.

The same symmetrical situation has been seen for $c < d$ and $c > d$ theoretically and empirically. The worst-case analysis of **Binomial-stat** and **Fibonacci-stat** is consistent with the simulation results: the later performs well when computations and communications overlap. However, the worst-case analysis has been unable to highlight the probabilistic advantage of **Tree-dyn** over **Binomial-stat**. Therefore, this theoretical study is ineffective for assessing the effect of heterogeneity between methods, contrarily to the empirical study.

As future work, we plan to investigate more complex communication models, such as specific network topologies. It would also be interesting to design a better dynamic algorithm for non-commutative reductions, which avoids the situation when many processors are idle but cannot initiate a communication since no neighboring processors are free.

ACKNOWLEDGEMENT

We would like to thank the reviewers for their comments and suggestions, which greatly improved the final version of this paper. This work was supported in part by the ANR *RESCUE* and *SOLHAR* projects. A. Benoit is with the Institut Universitaire de France.

DOI: 10.1002/cpe

REFERENCES

1. Pjesivac-Grbovic J, Angskun T, Bosilca G, Fagg G, Gabriel E, Dongarra J. Performance analysis of MPI collective operations. *IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2005.
2. Thakur R, Rabenseifner R, Gropp W. Optimization of Collective communication operations in MPICH. *International Journal of High Performance Computing Applications* 2005; **19**:49–66.
3. Rabenseifner R. Optimization of Collective Reduction Operations. *Computational Science - ICCS 2004*, *Lecture Notes in Computer Science*, vol. 3036, Bubak M, van Albada G, Sloot P, Dongarra J (eds.). Springer Berlin / Heidelberg, 2004; 1–9.
4. Liu P, Kuo MC, Wang DW. An Approximation Algorithm and Dynamic Programming for Reduction in Heterogeneous Environments. *Algorithmica* Feb 2009; **53**(3):425–453.
5. Legrand A, Marchal L, Robert Y. Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms. *Journal of Parallel and Distributed Computing* 2005; **65**(12):1497–1514.
6. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM* 2008; **51**(1):107–113.
7. Zaharia M, Konwinski A, Joseph A, Katz R, Stoica I. Improving MapReduce performance in heterogeneous environments. *Proc. of the 8th USENIX conf. on Operating systems design and implementation*, 2008; 29–42.
8. Bar-Noy A, Kipnis S, Schieber B. An optimal algorithm for computing census functions in message-passing systems. *Parallel Processing Letters* 1993; **3**(1):19–23.
9. Bruck J, Ho CT. Efficient global combine operations in multi-port message-passing systems. *Parallel Processing Letters* 1993; **3**(4):335–346.
10. van de Geijn RA. On global combine operations. *Journal of Parallel and Distributed Computing* 1994; **22**(2):324–328.
11. Ali Q, Pai VS, Midkiff SP. Advanced collective communication in Aspen. *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '08, New York, NY, USA, 2008; 83–93.
12. Ritzdorf H, Träff JL. Collective operations in NEC's high-performance MPI libraries. *IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2006.
13. Bar-Noy A, Bruck J, Ho CT, Kipnis S, Schieber B. Computing global combine operations in the multiport postal model. *IEEE Transactions on Parallel and Distributed Systems* Aug 1995; **6**(8):896–900.
14. Rabenseifner R, Träff JL. More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Lecture Notes in Computer Science, Springer Berlin, 2004.
15. Chan EW, Heimlich MF, Purkayastha A, van de Geijn RA. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 2007; **19**(13):1749–1783.
16. Sanders P, Speck J, Träff JL. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Computing* 2009; **35**(12):581–594.
17. Kielmann T, Hofman RFH, Bal HE, Plaat A, Bhoedjang RAF. MPI's reduction operations in clustered wide area systems 1999.
18. Agarwal A, Chapelle O, Dudík M, Langford J. A Reliable Effective Terascale Linear Learning System. *CoRR* 2011; **abs/1110.4198**.
19. Canon LC, Jeannot E, Sakellariou R, Zheng W. Comparative Evaluation of the Robustness of DAG Scheduling Heuristics. *Proceedings of CoreGRID Integration Workshop*, Heraklion-Crete, Greece, 2008.
20. Benoit A, Dufossé F, Gallet M, Robert Y, Gaujal B. Computing the throughput of probabilistic and replicated streaming applications. *Proc. of SPAA, Symp. on Parallelism in Algorithms and Architectures*, 2010; 166–175.
21. Canon LC, Jeannot E. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments. *IEEE Trans. Parallel Distrib. Syst.* 2010; **21**(4):532–546.
22. Cordasco G, Chiara RD, Rosenberg AL. On scheduling DAGs for volatile computing platforms: Area-maximizing schedules. *J. Parallel Distrib. Comput.* 2012; **72**(10):1347–1360.
23. Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to Algorithms*. 3rd edn., The MIT Press, 2009.
24. Canon LC. Scheduling associative reductions with homogeneous costs when overlapping communications and computations. *IEEE International Conference on High Performance Computing (HiPC)*, 2013.
25. Lublin U, Feitelson DG. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comp.* 2003; **63**(11):1105–1122.
26. Feitelson D. Workload modeling for computer systems performance evaluation. *Book Draft, Version 0.38* 2013; .

DOI: 10.1002/cpe